Ticket controller

La classe **TicketController** si occupa della gestione delle richieste inerenti la visualizzazione, l’emissione e l’invalidazione dei biglietti. Le richieste che provengono dai client vengono valutate e sanificate per poi essere eventualmente inoltrate allo smart contract che gira sulla blockchain o al database.

La classe fornisce una serie di metodi pubblici:

* Il metodo **getTicket()** interroga il database per ottenere tutti i dati relativi ad un determinato biglietto (dati del possessore, dati dell’evento, sigillo fiscale, orario di emissione, …). Il metodo invoca la funzione **findOne()**, definita all’interno della libreria interna dell’applicazione web (cartella functions), per trovare il record associato al biglietto cercato in base al suo **id**. Il metodo verifica poi se l’utente che richiede di visualizzare i dati del biglietto sia anche colui che lo ha acquistato. In caso negativo la richiesta fallisce. Vengono eseguite altre due interrogazioni al database, stavolta per ottenere i dati relativi all’evento associato al biglietto e quelli relativi al suo possessore. Tutte e tre le interrogazioni al database restituiscono come risultato un oggetto JSON. I tre JSON vengono fusi insieme ottenendo una descrizione più dettagliata del biglietto. La funzione richiede come parametri:
  + L’**id dell’utente** che effettua la richiesta di interrogare il database;
  + L’**id del biglietto** di cui si vogliono ottenere i dati.

La funzione restituisce come risultato una **tupla** (array) contenente lo stato della richiesta (**codice HTTP 200** in caso di successo) e un **JSON** contenente i dati del biglietto. Se la richiesta fallisce viene restituito lo stato di fallimento (**codice HTTP 500**) e una stringa contenente un messaggio di errore.

* Il metodo **getTicketsByUser()** interroga il database per ottenere tutti i biglietti acquistati da un dato utente. Il metodo invoca la funzione **find()**, definita all’interno della libreria interna dell’applicazione web (cartella functions), per trovare tutti i record associati ad un certo **user id** (codice univoco assegnato all’utente che possiede il biglietto). La lista di record ottenuta viene poi filtrata scartando i record associati a biglietti già invalidati (campo **isUsed** con valore **true**). Viene eseguita un’altra interrogazione al database, stavolta per ottenere i dati relativi all’evento associato al biglietto. Tutte e due le interrogazioni al database restituiscono come risultato un oggetto JSON. I due JSON vengono fusi insieme ottenendo una descrizione più dettagliata del biglietto. La funzione richiede come parametro l’**id dell’utente** che effettua la richiesta di interrogare il database. La funzione restituisce come risultato una **tupla** (array) contenente lo stato della richiesta (**codice HTTP 200** in caso di successo) e un **lista** (array) contenente tutti i biglietti posseduti dall’utente. Se la richiesta fallisce viene restituito lo stato di fallimento (**codice HTTP 500**) e una stringa contenente un messaggio di errore.
* Il metodo **getTicketsByEvent()** interroga il database per ottenere tutti i biglietti emessi per un certo evento. Il metodo invoca la funzione **find()**, definita all’interno della libreria interna dell’applicazione web (cartella functions), per trovare tutti i record associati ad un certo **event id** (codice univoco assegnato agli eventi disponibili sul sito). La funzione richiede come parametro l’**id dell’evento** per cui si vuole ottenere un’elenco di tutti i biglietti emessi. La funzione restituisce come risultato una **tupla** (array) contenente lo stato della richiesta (**codice HTTP 200** in caso di successo) e un **lista** (array) contenente tutti i biglietti emessi per l’evento considerato. Se la richiesta fallisce viene restituito lo stato di fallimento (**codice HTTP 500**) e una stringa contenente un messaggio di errore.
* Il metodo **getInvalidatedTicketsByEvent()** è identico a **getTicketsByEvent()**, ma prima di restituire il risultato filtra l’elenco dei biglietti scartando quelli che non sono stati ancora invalidati. La lista ottenuta ci da informazioni sugli ingressi ad un evento.
* Il metodo **richiestaBiglietto()** controlla la disponibilità dei biglietti richiesti dall’utente sulla blockchain, poi genera una richiesta di pagamento che viene inoltrata al servizio PayPal. Se l’utente seleziona un metodo di pagamento diverso da PayPal, viene restituito il **codice** **HTTP 406** insieme ad un messaggio d’errore. Se sulla blockchain non è disponibile alcun **token** da emettere (numero posti disponibili minore del numero di biglietti richiesti dal cliente), la transazione per verificare la disponibilità dei biglietti fallisce e viene restituito il **codice** **HTTP 500** insieme ad un messaggio d’errore. Lo stesso codice viene restituito anche quando fallisce l’inoltro della richiesta di pagamento a PayPal. In caso di successo viene restituito il **codice HTTP 200** insieme al link PayPal su cui eseguire il pagamento. Il metodo richiede come parametri:
  + il **metodo di pagamento** selezionato dal cliente (PayPal, Visa, Mastercard);
  + l’**id dell’utente** che vuole acquistare i biglietti;
  + l**’id dell’evento** per cui si vogliono acquistare i biglietti;
  + il **numero di biglietti** che il cliente vuole acquistare i biglietti;
  + l’**indirizzo del wallet** del cliente che vuole acquistare i biglietti;
  + la **password del wallet** del cliente che vuole acquistare i biglietti.
* Il metodo **emissioneBiglietti()** esegue la transazione di acquisto addebitando il costo dei biglietti dal conto PayPal del cliente, emette i biglietti associati alla richiesta d’acquisto e invia una mail all’utente contenente la ricevuta di pagamento. L’emissione dei biglietti consiste di più passi:
  + Viene creato un **nuovo record** sul database per memorizzare i dati del biglietto;
  + Viene emesso un **token**, che ricoprirà il ruolo di gemello digitale del biglietto cartaceo;
  + Il **codice univoco** associato al token viene inserito nel record del database;
  + Viene aggiunto un **sigillo fiscale** al record del database;
  + Viene aggiunto un **QR code** contenente i dati del biglietto al record del database;
  + Viene decremento il **numero di posti disponibili** all’evento (sul database);
  + Il record viene effettivamente salvato sul database.

Se l’utente che richiede l’emissione dei biglietti non è lo stesso che ha effettuato lo richiesta di pagamento, viene restituito il **codice** **HTTP 401** insieme ad un messaggio d’errore. Se l’emissione del **token** sulla blockchain fallisce, viene restituito il **codice** **HTTP 401** insieme ad un messaggio d’errore. In caso di successo viene restituito il **codice HTTP 200** insieme ad un JSON contenente i dati della ricevuta di pagamento. Il metodo richiede come parametri:

* + l’**id PayPal dell’utente** che ha acquistato i biglietti;
  + l**’id del pagamento** effettuato dal cliente;
  + l’**email** del cliente (la ricevuta di pagamento viene inviata tramite mail);
  + l’**id dell’utente** che ha acquistato i biglietti;
  + l’**indirizzo del wallet** del cliente che ha acquistato i biglietti.
* Il metodo **invalidaBiglietto()** verifica l’integrità e l’autenticità del sigillo fiscale ed invoca il metodo dello smart contract che si occupa dell’invalidazione del **token** associato al biglietto. Terminata l’invalidazione viene aggiornato il database aggiungendo l’orario e la data di invalidazione del biglietto. Se il sigillo fiscale non è autentico o l’invalidazione del **token** sulla blockchain fallisce, viene restituito il **codice** **HTTP 500** insieme ad un messaggio d’errore. In caso di successo viene restituito il **codice HTTP 200** insieme ad un JSON contenente i dati del biglietto invalidato. Il metodo richiede come parametri:
  + i **dati del biglietto** codificati all’interno del QR code;
  + l’**indirizzo del wallet** dell’invalidatore dei biglietti;
  + la **password del wallet** dell’invalidatore dei biglietti.